Texas Hold'em Poker Game Simulator

OUR GROUP:

01/ Куцюк Даніїл
02/ Деркач Єлизавета
03/ Пархоменко Георгій
04/ Маслов Петро

STRUCTURE:

Функції:

- initialize_deck(): Ініціалізація колоди карт.
- shuffle_deck(): Тасування карт.
- deal_cards(): Роздача карт гравцю та супротивнику.
- calculate_win_probability(): Обчислення ймовірності виграшу.
- is_hand_better(): Порівняння рук гравця і супротивника.

Головний цикл:

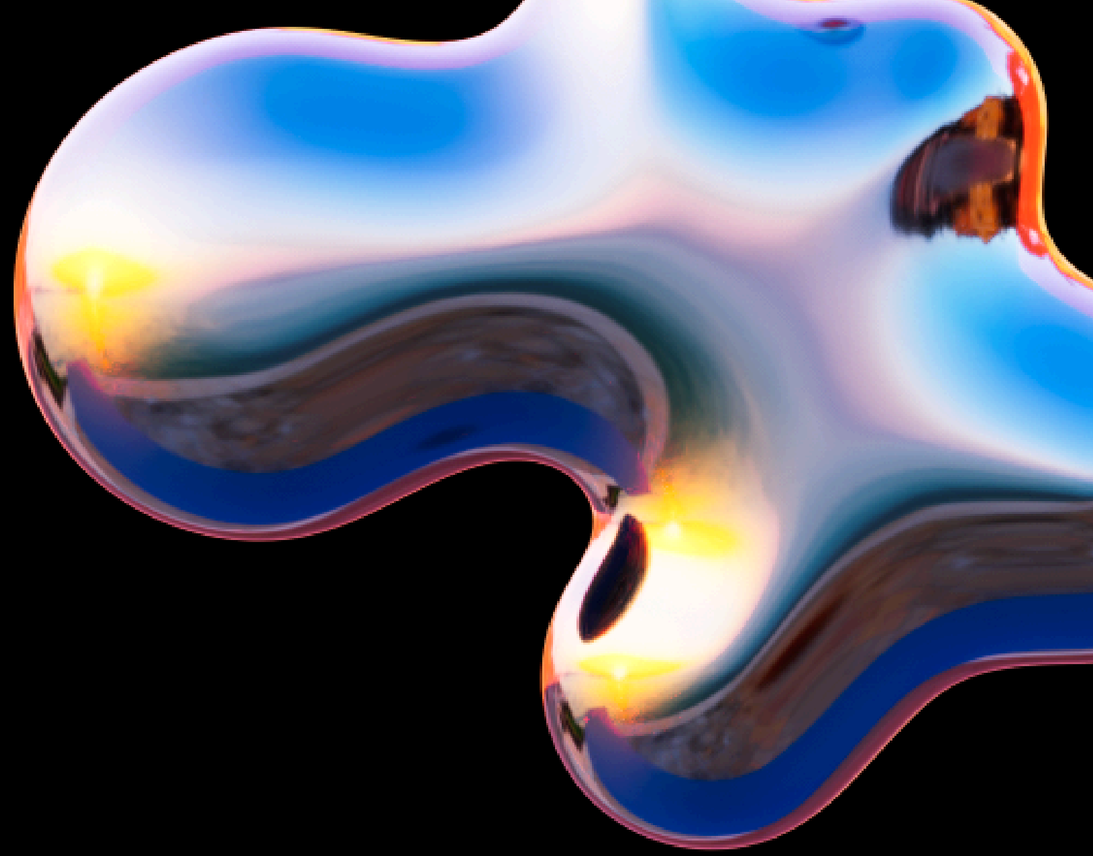
- Обробка дій гравця (скидання або продовження).
- Додавання нових карт на стіл (флоп, терн, рівер).
- Підрахунок остаточної ймовірності виграшу.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>

#define NUM_CARDS 52
#define NUM_RANKS 13
#define NUM_SUITS 4
#define HAND_SIZE 2
#define FLOP_SIZE 3
#define TURN_RIVER_SIZE 1
#define MONTE_CARLO_SIMULATIONS 10000

const char *ranks[] = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};
const char *suits[] = {"(H)", "(D)", "(C)", "(S)"};
```

Ця частина коду визначає основні константи та ініціалізує масиви рангу та масті карт. Кількість карт у колоді фіксована, і масиви 'ranks' та 'suits' допомагають у відображенні ігрових карт.

Часова складність: O(1) для доступу до елементів масиву.
Просторова складність: O(1) для зберігання цих значень.

```
typedef struct {
  int rank;
  int suit;
} Card;

typedef struct {
  int combination_level; // Рівень комбінації, чим вище, тим сильніша комбінація
  int highest_card;    // Старша карта в комбінації
  int second_highest_card; // Друга старша карта (наприклад, для двох пар)
} HandEvaluation;
```



Структура 'Card' зберігає дані про кожну карту (ранг та масть). Структура 'HandEvaluation' використовується для оцінки сили руки гравця.

Часова складність: O(1) для доступу до полів структури. Просторова складність: O(1) для кожної структури.

```
void shuffle_deck(Card *deck) {
        for (int i = num_cards - 1; i > 0; i--) {
            int j = rand() % (i + 1);
            Card temp = deck[i]
            deck[i] = deck[j];
            deck[j] = temp;
        }
}
```

Алгоритм перемішування Fisher-Yates використовується для випадкового перетасування карт у колоді. Цей алгоритм гарантує рівномірне перемішування карт без повторів.

Часова складність: O(n), де n — кількість карт у колоді.
Просторова складність: O(1), так як всі операції відбуваються на місці.

```
void deal_cards(Card *deck, int *current_card, Card *hand, int size) {
    for (int i = 0; i < size; i++) {
        hand[i] = deck[*current_card];
        (*current_card)++;
    }
}
```

Функція роздачі карт 'deal_cards' бере карти з колоди і додає їх у руку гравця. Це дозволяє уникнути повторного роздавання тих самих карт за рахунок використання 'current_card'.

Часова складність: O(k), де k — кількість карт, які треба роздати.
Просторова складність: O(k) для збереження карт.

```
int compare_cards(const void *a, const void *b) {
    Card *cardA = (Card *)a;
    Card *cardB = (Card *)b;
    return cardB->rank - cardA->rank;
}
```

Функція 'compare_cards' використовується для сортування карт за рангом, що є важливим етапом під час оцінки сили комбінацій.

Часова складність: O(1) для порівняння двох карт.
Просторова складність: O(1), оскільки порівняння відбувається без додаткового використання пам'яті.

```
HandEvaluation evaluate_hand(Card *hand, Card *community_cards, int community_size) {
    Card all_cards[HAND_SIZE + community_size];
      for (int i = 0; i < HAND_SIZE; i++) {
        all_cards[i] = hand[i];
  }
    for (int i = 0; i < community_size; i++) {
       all_cards[HAND_SIZE + i] = community_cards[i];
  }
  qsort(all_cards, HAND_SIZE + community_size, sizeof(Card), compare_cards);
  HandEvaluation eval = {0, all_cards[0].rank, -1};
  return eval;
}
```

Функція 'evaluate_hand' аналізує руку гравця та спільні карти на столі. Використовується функція сортування 'qsort' для впорядкування карт за рангом. Це дозволяє оцінити, які комбінації є у гравця, такі як пари, стріти, флеші тощо.

Часова складність: O(n log n) для сортування n карт.
Просторова складність: O(n) для збереження усіх карт.

```
bool is_hand_better(Card *hand1, Card *hand2, Card *community_cards, int
community_size) {
  HandEvaluation eval1 = evaluate_hand(hand1, community_cards,
community_size);
  HandEvaluation eval2 = evaluate_hand(hand2, community_cards,
community_size);
  if (eval1.combination_level > eval2.combination_level) return true;
  if (eval1.combination_level < eval2.combination_level) return false;
  return eval1.highest_card > eval2.highest_card;
 }
```

Функція 'is_hand_better' порівнює дві руки гравців і визначає, яка з них сильніша. Оцінка відбувається за рівнем комбінації та старшими картами. Це ключова функція для визначення переможця гри.

Часова складність: O(n log n), оскільки для кожної руки відбувається сортування карт.
Просторова складність: O(n) для збереження та порівняння результатів.

```
int calculate_win_probability(Card *player_hand, Card *community_cards, int community_size) {
  int wins = 0, ties = 0, losses = 0;
  Card opponent_hand[HAND_SIZE], deck[NUM_CARDS];
  int current_card;

  for (int i = 0; i < MONTE_CARLO_SIMULATIONS; i++) {
    initialize_deck(deck);
    shuffle_deck(deck);
    deal_cards(deck, &current_card, opponent_hand, HAND_SIZE);

    if (is_hand_better(player_hand, opponent_hand, community_cards, community_size)) wins++;
    else if (is_hand_better(opponent_hand, player_hand, community_cards, community_size)) losses++;
    else ties++;
  }
  return (wins * 100) / (wins + losses + ties);
}
```

Функція 'calculate_win_probability' використовує метод Монте-Карло для оцінки ймовірності виграшу руки гравця. Ця симуляція повторюється багато разів для точнішого результату. Чим більше симуляцій, тим точніша оцінка.

Часова складність: O(k * n log n), де k – кількість симуляцій, а n – кількість карт у роздачі. Просторова складність: O(n) для збереження колоди та рук.

```
int calculate_win_probability(Card *player_hand, Card *community_cards, int community_size) {
    int wins = 0;
    int ties = 0;
    int losses = 0;
    Card opponent_hand[HAND_SIZE];
    Card deck[NUM_CARDS];
    int current_card;

    for (int i = 0; i < MONTE_CARLO_SIMULATIONS; i++) {
        initialize_deck(deck);
        shuffle_deck(deck);
        current_card = 0;

        // Remove player and community cards from the deck
        for (int j = 0; j < HAND_SIZE; j++) {
            deck[player_hand[j].rank * NUM_SUITS + player_hand[j].suit] = deck[--num_cards];
        }
        for (int j = 0; j < community_size; j++) {
            deck[community_cards[j].rank * NUM_SUITS + community_cards[j].suit] = deck[--num_cards];
        }

        // Deal opponent's hand
        deal_cards(deck, &current_card, opponent_hand, HAND_SIZE);

        // Compare hands
        if (is_hand_better(player_hand, opponent_hand, community_cards, community_size)) {
        wins++;
        } else if (is_hand_better(opponent_hand, player_hand, community_cards, community_size)) {
            losses++;
        } else {
            ties++;
        }
    }

    return (wins * 100) / (wins + losses + ties);
}
```

Функція calculate_win_probability виконує Монте-Карло симуляції для визначення ймовірності виграшу руки гравця. Вона імітує тисячі ігор проти випадкових рук опонентів і визначає ймовірність перемоги, програшу або нічиї.

Часова складність: O(M * n log n), де M – кількість симуляцій, а n – кількість карт у роздачі.
Просторова складність: O(n), де n – кількість карт у колоді.

```c
int main() {
  srand(time(NULL));

  Card deck[NUM_CARDS];
  initialize_deck(deck);
  shuffle_deck(deck);

  int current_card = 0;

  Card player_hand[HAND_SIZE];
  Card opponent_hand[HAND_SIZE];
  Card community_cards[FLOP_SIZE + TURN_RIVER_SIZE * 2];

  // Deal player and opponent hands
  deal_cards(deck, &current_card, player_hand, HAND_SIZE);
  deal_cards(deck, &current_card, opponent_hand, HAND_SIZE);

  // Preflop win probability
  int win_probability = calculate_win_probability(player_hand, NULL, 0);
  printf("** Preflop Probability of Winning **\n");
  printf("Your preflop probability of winning is: %d%%\n", win_probability);

  // Deal the flop
  deal_cards(deck, &current_card, community_cards, FLOP_SIZE);

  // Print player hand
  printf("** Your Hand **\n");
  for (int i = 0; i < HAND_SIZE; i++) {
    print_card(player_hand[i]);
    printf(" ");
  }
  printf("\n");

  // Print community cards (flop)
  printf("** Community Cards (Flop) **\n");
  for (int i = 0; i < FLOP_SIZE; i++) {
    print_card(community_cards[i]);
    printf(" ");
  }
  printf("\n");

  // Calculate and print win probability
  win_probability = calculate_win_probability(player_hand, community_cards, FLOP_SIZE);
  printf("** Probability of Winning **\n");
  printf("Your probability of winning is: %d%%\n", win_probability);

  return 0;
}
```

Функція main виконує основні кроки гри: роздає карти, виводить їх на екран, обчислює ймовірність перемоги та виводить результат симуляцій. Вона також включає функцію calculate_win_probability для оцінки шансів гравця на перемогу на різних етапах гри.

Часова складність: O(M * n log n) для симуляцій та обчислення ймовірності виграшу, де M – кількість симуляцій, а n – кількість карт.
Просторова складність: O(n) для зберігання колоди, рук та комбінацій.

COMPILATION

1. Відкрийте термінал і перейдіть до директорії, де збережений файл poker.c.
2. Використовуйте наступну команду для компіляції програми:

   gcc poker.c -o poker

- Після успішної компіляції запустіть програму командою:

  Для Linux/MacOS: ./poker

  Для Windows: poker.exe

```
** Preflop Probability of Winning **Your preflop probability of winning is: 43%*
* Your Hand **
[ 8(H) ] [ J(H) ]
** Community Cards (Flop) **
[ 3(S) ] [ 6(C) ] [ J(S) ]
** Probability of Winning **
Your probability of winning is: 74%
Enter your action (F/C): c
** Community Cards (Turn)
[ 3(S) ] [ 6(C) ] [ J(S) ] [ Q(S) ]
 Updated Probability of Winning
Your updated probability of winning is: 59%
Enter your action (F/C): c
 Community Cards (River)
[ 3(S) ] [ 6(C) ] [ J(S) ] [ Q(S) ] [ 2(C) ]
 Final Probability of Winning
Your final probability of winning is: 53%
 Opponent's Hand
[ 6(S) ] [ 4(D) ]
 Winner of the Game **
Player wins
```

```
** Preflop Probability of Winning **Your preflop probability of winning is: 25%*
* Your Hand **
[ 9(C) ] [ 3(C) ]
** Community Cards (Flop) **
[ 9(S) ] [ 10(C) ] [ K(C) ]
** Probability of Winning **
Your probability of winning is: 56%
Enter your action (F/C): c
** Community Cards (Turn)
[ 9(S) ] [ 10(C) ] [ K(C) ] [ Q(S) ]
 Updated Probability of Winning
Your updated probability of winning is: 37%
Enter your action (F/C): c
 Community Cards (River)
[ 9(S) ] [ 10(C) ] [ K(C) ] [ Q(S) ] [ J(H) ]
 Final Probability of Winning
Your final probability of winning is: 30%
 Opponent's Hand
[ 10(S) ] [ 2(C) ]
 Winner of the Game **
Opponent wins
```

Thanks